

HOW TO... Master macros

In the second part of our guide to automating tasks in Excel, Lynley Oram explains how to get to grips with Visual Basic for Applications and edit macro code

Macros can help to automate many of the repetitive tasks you undertake in Excel. Last month, we looked at how to record a macro using the built-in Macro Recorder. A copy of that article is included on this month's cover disc. In this article, we will delve deeper into macros. In Excel, macros are written in a language called Visual Basic for Applications (VBA). Even macros created using the Macro Recorder will exist as VBA code within Excel.

To Excel, a macro is a procedure. Procedures are VBA statements that perform tasks or return a result. There are two types of procedure you can create in Excel: subroutines and functions.

Functions return a result. If you've been using Excel for a while, you'll already have used functions, even if it is just the simple SUM function to add up a column of figures. The code for a function starts with Function and finishes with End Function. Then there are subroutines, the category under which macros usually fall. A subroutine means that the code performs a specific task. In VBA, a subroutine's code starts with Sub and finishes with the words End Sub.

Here, we'll show you how to use VBA.

1 Macro code is edited in the Visual Basic Editor (VBE). You can open this by going to the Tools menu and selecting Macro, Visual Basic Editor, or simply pressing Alt-F11. The window that opens is divided into panes. There's a list of VBAProjects on the left; this is the Project

Explorer. The project name will be the name of the workbook in which the macro is held. Expand one of the projects by clicking on the '+' symbol next to it and then on a module listed under the project. The code for that module will appear in the right-hand pane, the Code window.

It's also useful to have the Properties window visible. Click on the View menu and select Properties Window. This pane will appear below the Project Explorer. For now, close the Editor.

2 To get started, we're going to play with the VBA code for the macro that we created last month. If you saved this macro in the personal.xls workbook, you'll need to unhide the workbook before you start editing the macro. Go to the Windows menu and select Unhide. In the dialog box that appears, click on personal.xls to select it and then click OK. Use Windows, Hide when you are finished – otherwise, the Personal workbook will continue to open each time you start Excel.

3 In the VBE, open the macro we created last month. Go to Tools, Macro, Macros. Click on the relevant macro in the box that appears, and then on Edit. You'll see the code for this macro in the pane on the right. A macro is made up of VBA statements, and you can work out what many of these are just by looking at them. Literal text is always enclosed between double quotation marks. It is important that you don't

delete these marks, because without them the macro probably won't run.

We are going to make a simple edit. For example, let's say you want the macro to print three copies of your report instead of two. You need to change this part of the code:

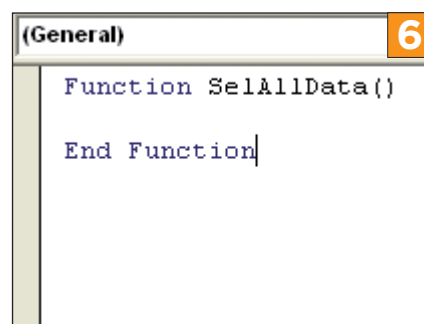
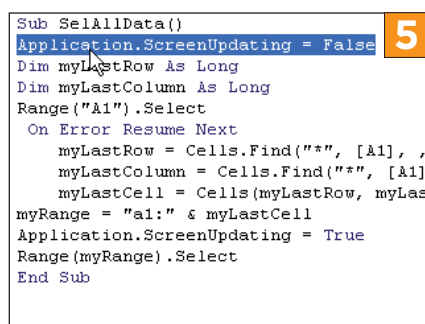
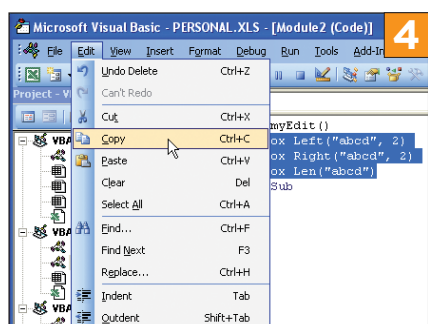
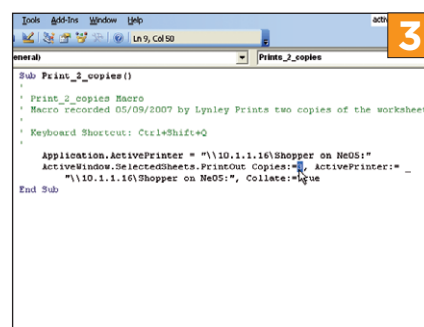
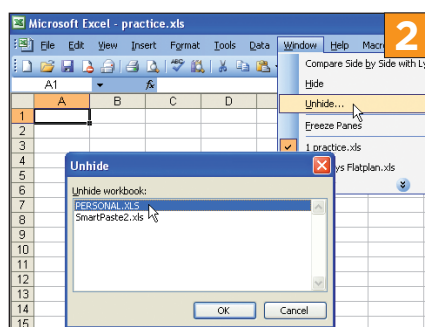
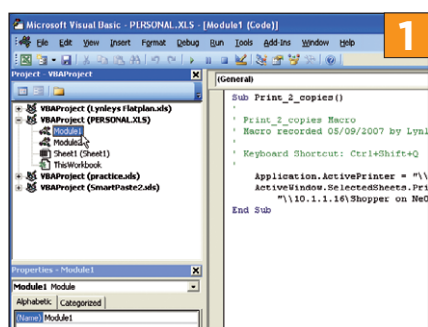
Copies:=2

so it reads:

Copies:=3

This is a lot easier than deleting the macro and recording a new one. You can use this method to make simple changes to macros, such as changing the names the macro might be required to insert in a workbook.

4 If you need to make a substantial change to a complex macro, try this technique. Instead of re-recording the entire macro, create a new macro for just the part that has changed. Open that macro in the VBE and highlight all the code between the Sub and the End Sub lines. Copy and paste that code into the Code window for the original macro. You should be able to work out from the VBA statements where the new code should be placed. If you are inserting it, make sure you put it on a new blank line. If it is replacing existing code, be absolutely sure you're deleting the right portion.



5 As well as using the VBE to alter what the macro does, you can use it to change the way the macro works. There is a handy edit you can make to many macros to make them run faster. Say your macro is making changes on a row-by-row basis and you've got hundreds or thousands of rows. Here's how to alter the macro so you don't have to see each change as it is made. Open the macro. The code for the macro will start with Sub, then the macro's name. So in the macro mentioned in Step 3, the code starts with:

```
Sub Print_2_copies()
```

Print_2_copies is the macro's name. Directly under the 'Sub your macro name' line, enter:

```
Application.ScreenUpdating = False
```

6 To get started writing a macro from scratch, go to Tools, Macro, Macros. Enter a name for the macro and select the appropriate option from the Macros In menu at the bottom of the dialog box. Click Create. This opens the Visual Basic Editor. In the Code window, three lines will have been entered for you. The first line contains the word Sub followed by the macro name and parentheses, the second line is blank and the third line contains the words End Sub. The cursor will be flashing at the start of the second, blank line ready for you to enter the VBA statements. If you want to create a function rather than a subroutine, change the first Sub to Function. The last line will change automatically to End Function.

7 Enter all VBA statements in lower case. When you've entered a line of code and move on to the next line, the VBE will capitalise at least one letter in each word, if that word has been entered correctly. It will also capitalise the first letter of variables and constants in the statement. If this doesn't happen, you will immediately know that something is amiss, and you should check your code straight away.

```
Sub SelAllData()  
Application.ScreenUpdating = False  
Dim myLastRow As Long  
Dim myLastColumn As Long  
Range("A1").Select  
On Error Resume Next  
myLastRow = Cells.Find("A", [myLastCell],  
myLastColumn = Cells.Find("A", [myLastCell],  
myrange = "A1:" & myLastCell
```

```
Sub ATest1()  
Application.ScreenUpdating = False  
Do Until ActiveCell.Value = ""  
If ActiveCell.Value = 0 Then  
ActiveCell.EntireRow.Delete  
Else  
ActiveCell.Offset(1, 0).Select  
End If  
Loop  
End Sub
```

10 Microsoft Visual Basic Help

Do...Loop Statement

See Also Example Specifics

Repeats a block of statements while a condition becomes True.

Syntax

Do {While | Until} condition
[statements]
[Exit Do]
[statements]
Loop

Or, you can use this syntax:

8 To see how this works, let's try writing a macro that will delete all the rows in a workbook that have a value of zero. Go to Tools, Macro, Macros. Enter ATest1 as the name. The name must start with a letter and it cannot contain spaces. You can use only letters, numbers and the underscore character. Click Create. Enter this code between the Sub and End Sub lines:

```
Application.ScreenUpdating = False  
Do Until ActiveCell.Value = ""  
If ActiveCell.Value = 0 Then  
ActiveCell.EntireRow.Delete  
Else  
ActiveCell.Offset(1, 0).Select  
End If  
Loop
```

9 Line two in the code you entered in Step 8 starts the procedure looking for rows in the workbook that do not contain a value. The third line checks this value to see if it is a zero. If it is, then the code in line four will delete that row. If it doesn't, as indicated by the Else in line five, the program moves to line six, where it is told to move to the next row. That's the end of the decision process, as indicated by line seven. Line eight in the code will start the whole process again.

The macro will continue to loop though all the rows in the workbook until it reaches the end. The first line in the code ensures you don't have to see the macro doing this row by row, which is handy if you have hundreds of rows entered in your workbook. Click on the Save icon on the toolbar at the top of the VBE window.

10 Get in the habit of making good use of the VBA help files. For example, click inside the word Loop at the end of the code for this macro and press the F1 key on your keyboard. This will

FURTHER READING

VBA is an object-oriented programming language. For an overview of objects, methods, properties, events and collections, visit the Microsoft Knowledge Base (<http://tinyurl.com/yw6yg9>)

bring up the Help entry for Do ... Loop, showing you exactly what this statement is doing and how it works. You can do this for any word that you see used in VBA code.

The help files are an extremely powerful tool when writing VBA code. You will find them used quite often by even the most experienced programmers.

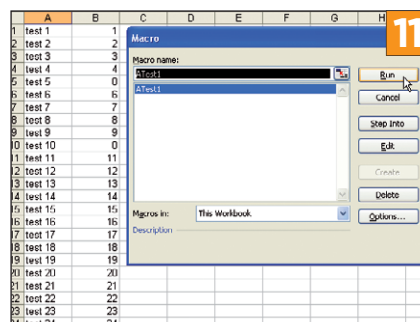
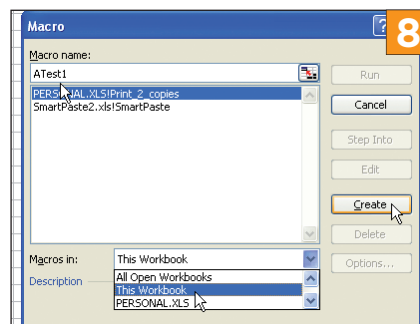
11 To try this macro out, open a new workbook. Enter Test 1, test 2 and so on down Column A. You just have to enter this into the first two rows. Then click on the Fill handle (the small black square in the bottom right of cell A2) and drag down the column to, say, row 30. Enter 1, then 2 in cells B1 and B2, then use the Fill handle to number the rows down to cell B30. Replace the contents of cell B5 with 0, and enter 0 in cell B10. Click on cell A1 and then select Tools, Macro, Macros. Highlight the ATest1 macro and click Run. Rows five and 10 will disappear.

12 You'll notice that, when looking at VBA code, some of the lines are indented. This is to make the code easier to read. Excel ignores indents when running the code. Indented lines indicate that they are all part of the same decision-making process (that is, the control structure). So, in the code for Step 8, all the lines that are part of our Loop to look for lines containing 0 are indented.

Another way to make code easier to read at a later date is to add comments. A comment is preceded by an apostrophe. Excel ignores lines that start with an apostrophe, so it won't affect the way the code is run. In the macro we created in Step 8, enter the following line anywhere:

```
'this is my first VBA code
```

Then click anywhere within the Code window. Excel colours the comment line green. 



```
Sub ATest1()  
Application.ScreenUpdating = False  
Do Until ActiveCell.Value = ""  
If ActiveCell.Value = 0 Then  
ActiveCell.EntireRow.Delete  
Else  
ActiveCell.Offset(1, 0).Select  
End If  
Loop  
End Sub
```

```
Sub ATest1()  
Application.ScreenUpdating = False  
'this is my first VBA code  
  
Do Until ActiveCell.Value = ""  
If ActiveCell.Value = 0 Then  
ActiveCell.EntireRow.Delete  
Else  
ActiveCell.Offset(1, 0).Select  
End If  
Loop  
End Sub
```